



Ravi-Raghavan /  
ConvexOptimization



<> Code

Issues

Pull requests

Actions

Projects

Security

Insights



ConvexOptimization / Homework #4 - Part 2.ipynb



Ravi-Raghavan Ready to submit

6 minutes ago



1271 lines (1271 loc) · 1.42 MB

# Homework #4 - Ravi Raghavan

```
In [31]: #Import Numpy and set the random seed number to 42
import numpy as np
np.random.seed(42)

#Import matplotlib
import matplotlib.pyplot as plt
```

## Vanilla Gradient Descent

```
In [32]: #Run Vanilla Gradient Descent Algorithm
#f: function that we are trying to optimize
#gradient: function that computes gradient of f at particular point
#x0: initial starting point
#alpha: fixed step size
#max_iter: maximum number of iterations to run
#epsilon: converge criterion for function value
def vanilla_gradient_descent(f, gradient, x0: np.ndarray, alpha, max_iter, epsilon):
    x = x0
    fx = f(x)

    #maintain arrays to store iterates and function values throughout gradient
    function_values = []
    function_values.append(fx)
    points = np.array([x])

    #enter for loop of max_iter times
    for iter in range(max_iter):
        grad = gradient(x) #compute gradient
        descent_direction = -1 * grad #our descent direction is the negative gradient
        x = x + (alpha * descent_direction) #compute next iterate
        points = np.append(points, x[np.newaxis, :, :], axis=0) #store point in array

        #if we have satisfied our convergence criteria, break from loop
        if np.abs(f(x) - fx) <= epsilon:
            break

        #calculate updated function value
        fx = f(x)
        function_values.append(fx)

    #store function value in array
    function_values = np.array(function_values)
    return points, function_values
```

## Exact Line Search

```
In [33]: #Run Gradient Descent Algorithm with Exact Line Search
#f: function that we are trying to optimize
#gradient: function that computes gradient of f at particular point
#exact_line_search: function to calculate the best step size to take at every iteration
#x0: initial starting point
#max_iter: maximum number of iterations to run
```

```

#max_iter: maximum number of iterations to run
#epsilon: converge criterion for function value
def exact_line_search_gradient_descent(f, gradient, exact_line_search, x0: np.ndarray):
    x = x0
    fx = f(x)

    #maintain arrays to store iterates and function values throughout gradient descent
    function_values = []
    function_values.append(fx)
    points = np.array([x])

    #enter for loop of max_iter times
    for iter in range(max_iter):
        grad = gradient(x) #compute gradient
        descent_direction = -1 * grad #our descent direction is the negative gradient
        alpha = exact_line_search(x, descent_direction) #compute best step size
        x = x + (alpha * descent_direction) #compute next iterate
        points = np.append(points, x[np.newaxis, :, :], axis=0) #store point in array

        #if we have satisfied our convergence criteria, break from loop
        if np.abs(f(x) - fx) <= epsilon:
            break

        #calculate updated function value
        fx = f(x)
        function_values.append(fx)

    #store function value in array
    function_values = np.array(function_values)
    return points, function_values

```

## Backtracking Line Search

```

In [34]: #Run Gradient Descent Algorithm with Exact Line Search
#f: function that we are trying to optimize
#gradient: function that computes gradient of f at particular point
#backtracking_algorithm: function to run the backtracking algorithm to get the best step size
#x0: initial starting point
#max_iter: maximum number of iterations to run
#epsilon: converge criterion for function value
def backtracking_line_search_gradient_descent(f, gradient, backtracking_algorithm, x0: np.ndarray):
    x = x0
    fx = f(x)

    #maintain arrays to store iterates and function values throughout gradient descent
    function_values = []
    function_values.append(fx)
    points = np.array([x])

    #enter for loop of max_iter times
    for iter in range(max_iter):
        grad = gradient(x) #compute gradient
        descent_direction = -1 * grad #our descent direction is the negative gradient
        alpha = backtracking_algorithm(x, descent_direction) #compute best step size
        x = x + (alpha * descent_direction) #compute next iterate
        points = np.append(points, x[np.newaxis, :, :], axis=0) #store point in array

        #if we have satisfied our convergence criteria, break from loop
        if np.abs(f(x) - fx) <= epsilon:
            break

        #calculate updated function value

```

```

#calculate updated function value
fx = f(x)
function_values.append(fx)

#store function value in array
function_values = np.array(function_values)
return points, function_values

```

## Quadratic Function

$$f_1(x) = \frac{1}{2}(x_1^2 + \gamma x_2^2)$$

$$\frac{\partial f}{\partial x_1} = x_1$$

$$\frac{\partial f}{\partial x_2} = \gamma x_2$$

$$\nabla f_1(x) = \begin{pmatrix} x_1 \\ \gamma x_2 \end{pmatrix}$$

Exact Line Search Derivation:

$$f_1(x + \alpha \Delta x) = \frac{1}{2}(x_1 + \alpha \Delta x_1)^2 + \frac{\gamma}{2}(x_2 + \alpha \Delta x_2)^2$$

$$\frac{d}{d\alpha} f_1(x + \alpha \Delta x) = \Delta x_1(x_1 + \alpha \Delta x_1) + \gamma(x_2 + \alpha \Delta x_2)(\Delta x_2)$$

Set this derivative to 0 and solve for  $\alpha$

$$\Delta x_1(x_1 + \alpha \Delta x_1) + \gamma(x_2 + \alpha \Delta x_2)(\Delta x_2) = 0$$

$$x_1 \Delta x_1 + \alpha(\Delta x_1)^2 + \gamma x_2 \Delta x_2 + \gamma \alpha(\Delta x_2)^2 = 0$$

$$\alpha * ((\Delta x_1)^2 + \gamma(\Delta x_2)^2) = -x_1 \Delta x_1 - \gamma x_2 \Delta x_2$$

To get the value of  $\alpha$ , we simply solve this equation!

$$\alpha = \frac{-x_1 \Delta x_1 - \gamma x_2 \Delta x_2}{(\Delta x_1)^2 + \gamma(\Delta x_2)^2}$$

In [35]:

```

#Calculate the value of the quadratic function at x and given a gamma value
def f1(x: np.ndarray, gamma = 10):
    x1_squared = x[0, 0] ** 2 #x1^2
    x2_squared = x[1, 0] ** 2 #x2^2
    return 0.5 * (x1_squared + (gamma * x2_squared))

#compute the gradient of the quadratic function at x and given a gamma value
def f1_gradient(x: np.ndarray, gamma = 10):
    gradient_vector = np.zeros(shape = x.shape)
    gradient_vector[0, 0] = x[0, 0]
    gradient_vector[1, 0] = gamma * x[1, 0]
    return gradient_vector

#Given the exact form for the line search for the quadratic function, output the
#x: current iterate
#delta_x: descent direction
#gamma: value of gamma
def f1_exact_line_search(x: np.ndarray, delta_x: np.ndarray, gamma = 10):
    x1 = x[0, 0]
    x2 = x[1, 0]

```

```
delta_x1 = delta_x[0, 0]
delta_x2 = delta_x[1, 0]
```

```
numerator = (-1 * x1 * delta_x1) + (-1 * gamma * x2 * delta_x2)
denominator = (delta_x1 ** 2) + (gamma * (delta_x2 ** 2))
```

```
return numerator / denominator
```

```
#Run the backtracking algorithm for the optimization problem involving the quad
#x: current iterate
```

```
#delta_x: descent direction
```

```
#alpha: value of alpha in backtracking algorithm
```

```
#beta: value of beta in backtracking algorithm
```

```
#gamma: value of gamma
```

```
def f1_backtracking_algorithm(x: np.ndarray, delta_x: np.ndarray, alpha = 0.25,
                             t = 1
                             while (f1(x + (t * delta_x))) > (f1(x) + (alpha * t * (f1_gradient(x).T @ d
                             t = beta * t
                             return t
```

```
In [36]: x0 = np.array([[10], [1]]) #Initial value of iterate
```

```
In [37]: #run vanilla gradient descent for f1(i.e. the quadratic function)
f1_vanilla_gd_iterates, f1_vanilla_gd_function_values = vanilla_gradient_descen
```

```
In [38]: #define ranges for x1 and x2
x1 = np.linspace(-12, 12, 2000)
x2 = np.linspace(-5, 5, 2000)

#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)
X1, X2 = np.meshgrid(x1, x2)

#function to calculate f1 value
#x1, x2: inputs
#gamma: function input
def f1_plot(x1, x2, gamma = 10):
    return 0.5 * ((x1 ** 2) + (gamma * (x2 ** 2)))

#given points on the contour grid, get function values
F = f1_plot(X1, X2)

#set figure size
plt.figure(figsize = (12, 12))

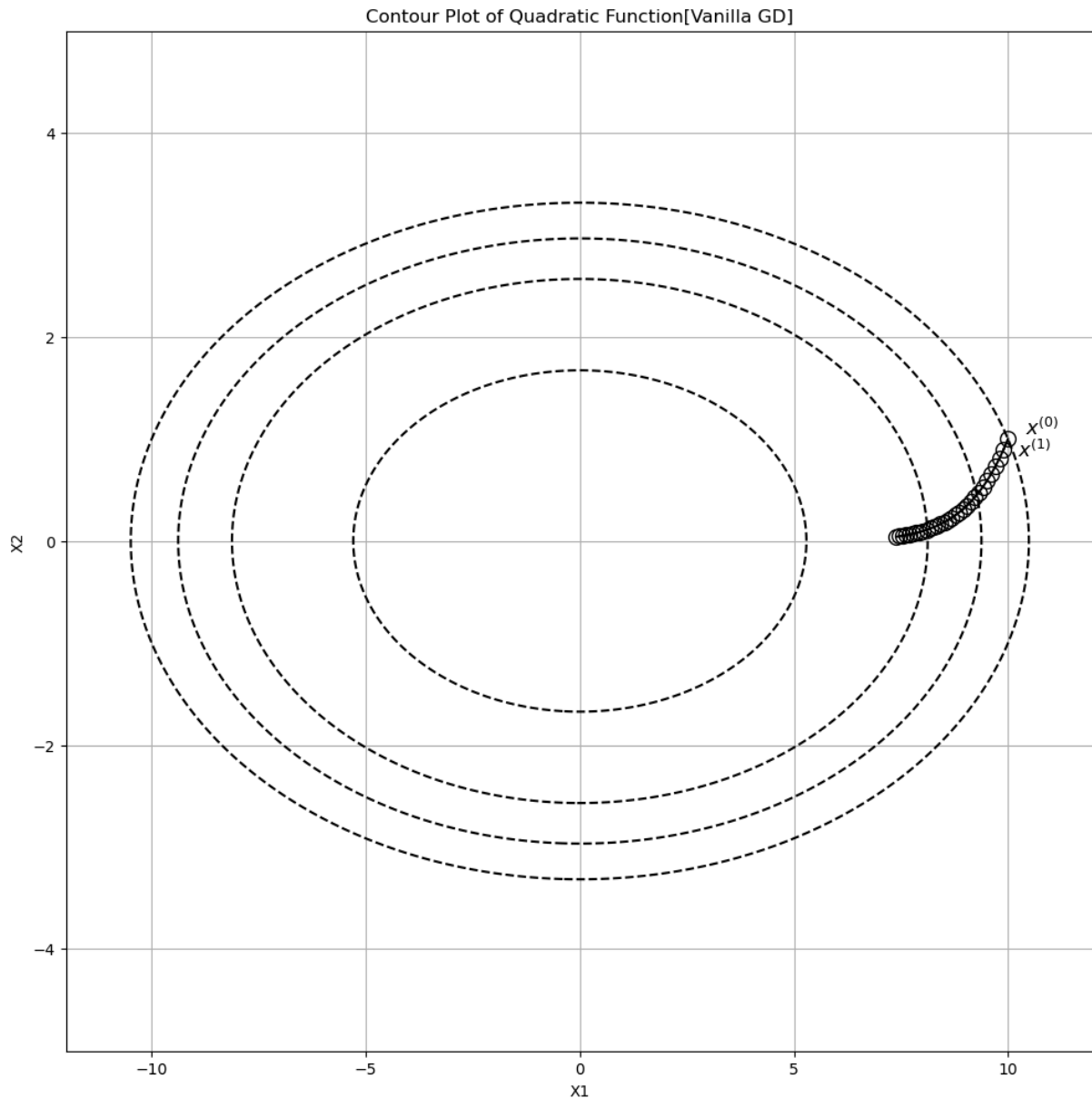
#plot f1 vanilla GD iterates
for point in f1_vanilla_gd_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label x^(0) and x^(1)
offset = 0.4
plt.text(f1_vanilla_gd_iterates[0][0, 0] + offset, f1_vanilla_gd_iterates[0][1,
plt.text(f1_vanilla_gd_iterates[1][0, 0] + 0.8 * offset, f1_vanilla_gd_iterates

#Connect iterates via a line
for i in range(len(f1_vanilla_gd_iterates)-1):
    plt.plot([f1_vanilla_gd_iterates[i][0, 0], f1_vanilla_gd_iterates[i+1][0, 0

#Plot contour lines
```

```
plt.contour(X1, X2, F, levels = [14, 33, 44, 55], colors='black', linestyle='d')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Quadratic Function[Vanilla GD]')
plt.grid(True)
plt.show()
```



```
In [39]: #run gradient descent, with exact line search, for f1(i.e. the quadratic function)
f1_exact_line_search_iterates, f1_exact_line_search_function_values = exact_line_search(f1, x0)
```

```
In [40]: #print iterates from exact line search for f1
f1_exact_line_search_iterates
```

```
Out[40]: array([[ 1.00000000e+01,
                 1.00000000e+00],
                [ 8.18181818e+00,
                 -8.18181818e-01],
                [ 6.69421488e+00,
                 6.69421488e-01],
                [ 5.54545455e+00,
                 -5.54545455e-01],
                [ 4.76190476e+00,
                 4.76190476e-01],
                [ 4.16666667e+00,
                 -4.16666667e-01],
                [ 3.70370370e+00,
                 3.70370370e-01],
                [ 3.33333333e+00,
                 -3.33333333e-01],
                [ 3.03030303e+00,
                 3.03030303e-01],
                [ 2.77777778e+00,
                 -2.77777778e-01],
                [ 2.55555556e+00,
                 2.55555556e-01],
                [ 2.36842105e+00,
                 -2.36842105e-01],
                [ 2.20576923e+00,
                 2.20576923e-01],
                [ 2.06349206e+00,
                 -2.06349206e-01],
                [ 1.93750000e+00,
                 1.93750000e-01],
                [ 1.82608696e+00,
                 -1.82608696e-01],
                [ 1.72727273e+00,
                 1.72727273e-01],
                [ 1.63956217e+00,
                 -1.63956217e-01],
                [ 1.56179775e+00,
                 1.56179775e-01],
                [ 1.49253731e+00,
                 -1.49253731e-01],
                [ 1.43137255e+00,
                 1.43137255e-01],
                [ 1.37796610e+00,
                 -1.37796610e-01],
                [ 1.33194444e+00,
                 1.33194444e-01],
                [ 1.29272609e+00,
                 -1.29272609e-01],
                [ 1.25992063e+00,
                 1.25992063e-01],
                [ 1.23305785e+00,
                 -1.23305785e-01],
                [ 1.21176471e+00,
                 1.21176471e-01],
                [ 1.19565217e+00,
                 -1.19565217e-01],
                [ 1.18438016e+00,
                 1.18438016e-01],
                [ 1.17777778e+00,
                 -1.17777778e-01],
                [ 1.17460317e+00,
                 1.17460317e-01],
                [ 1.17391304e+00,
                 -1.17391304e-01],
                [ 1.17500000e+00,
                 1.17500000e-01],
                [ 1.17727273e+00,
                 -1.17727273e-01],
                [ 1.18181818e+00,
                 1.18181818e-01],
                [ 1.18823529e+00,
                 -1.18823529e-01],
                [ 1.19625000e+00,
                 1.19625000e-01],
                [ 1.20576923e+00,
                 -1.20576923e-01],
                [ 1.21625000e+00,
                 1.21625000e-01],
                [ 1.22766667e+00,
                 -1.22766667e-01],
                [ 1.24000000e+00,
                 1.24000000e-01],
                [ 1.25319149e+00,
                 -1.25319149e-01],
                [ 1.26714286e+00,
                 1.26714286e-01],
                [ 1.28185714e+00,
                 -1.28185714e-01],
                [ 1.29723810e+00,
                 1.29723810e-01],
                [ 1.31328571e+00,
                 -1.31328571e-01],
                [ 1.32990000e+00,
                 1.32990000e-01],
                [ 1.34707692e+00,
                 -1.34707692e-01],
                [ 1.36471429e+00,
                 1.36471429e-01],
                [ 1.38270000e+00,
                 -1.38270000e-01],
                [ 1.40102564e+00,
                 1.40102564e-01],
                [ 1.41958182e+00,
                 -1.41958182e-01],
                [ 1.43835714e+00,
                 1.43835714e-01],
                [ 1.45733333e+00,
                 -1.45733333e-01],
                [ 1.47650000e+00,
                 1.47650000e-01],
                [ 1.49584615e+00,
                 -1.49584615e-01],
                [ 1.51535714e+00,
                 1.51535714e-01],
                [ 1.53501905e+00,
                 -1.53501905e-01],
                [ 1.55482857e+00,
                 1.55482857e-01],
                [ 1.57477273e+00,
                 -1.57477273e-01],
                [ 1.59484000e+00,
                 1.59484000e-01],
                [ 1.61492000e+00,
                 -1.61492000e-01],
                [ 1.63500000e+00,
                 1.63500000e-01],
                [ 1.65507692e+00,
                 -1.65507692e-01],
                [ 1.67525000e+00,
                 1.67525000e-01],
                [ 1.69550000e+00,
                 -1.69550000e-01],
                [ 1.71581250e+00,
                 1.71581250e-01],
                [ 1.73618750e+00,
                 -1.73618750e-01],
                [ 1.75671429e+00,
                 1.75671429e-01],
                [ 1.77728571e+00,
                 -1.77728571e-01],
                [ 1.79799048e+00,
                 1.79799048e-01],
                [ 1.81881250e+00,
                 -1.81881250e-01],
                [ 1.83974074e+00,
                 1.83974074e-01],
                [ 1.86076429e+00,
                 -1.86076429e-01],
                [ 1.88186250e+00,
                 1.88186250e-01],
                [ 1.90301429e+00,
                 -1.90301429e-01],
                [ 1.92421000e+00,
                 1.92421000e-01],
                [ 1.94543750e+00,
                 -1.94543750e-01],
                [ 1.96668571e+00,
                 1.96668571e-01],
                [ 1.98794286e+00,
                 -1.98794286e-01],
                [ 2.00929000e+00,
                 2.00929000e-01],
                [ 2.03071429e+00,
                 -2.03071429e-01],
                [ 2.05220000e+00,
                 2.05220000e-01],
                [ 2.07373571e+00,
                 -2.07373571e-01],
                [ 2.09531250e+00,
                 2.09531250e-01],
                [ 2.11692000e+00,
                 -2.11692000e-01],
                [ 2.13854688e+00,
                 2.13854688e-01],
                [ 2.16018125e+00,
                 -2.16018125e-01],
                [ 2.18181250e+00,
                 2.18181250e-01],
                [ 2.20342857e+00,
                 -2.20342857e-01],
                [ 2.22500000e+00,
                 2.22500000e-01],
                [ 2.24652500e+00,
                 -2.24652500e-01],
                [ 2.26800000e+00,
                 2.26800000e-01],
                [ 2.28941429e+00,
                 -2.28941429e-01],
                [ 2.31085714e+00,
                 2.31085714e-01],
                [ 2.33231250e+00,
                 -2.33231250e-01],
                [ 2.35377000e+00,
                 2.35377000e-01],
                [ 2.37522500e+00,
                 -2.37522500e-01],
                [ 2.39667500e+00,
                 2.39667500e-01],
                [ 2.41811667e+00,
                 -2.41811667e-01],
                [ 2.43954000e+00,
                 2.43954000e-01],
                [ 2.46093571e+00,
                 -2.46093571e-01],
                [ 2.48239000e+00,
                 2.48239000e-01],
                [ 2.50380000e+00,
                 -2.50380000e-01],
                [ 2.52515429e+00,
                 2.52515429e-01],
                [ 2.54654000e+00,
                 -2.54654000e-01],
                [ 2.56794500e+00,
                 2.56794500e-01],
                [ 2.58935714e+00,
                 -2.58935714e-01],
                [ 2.61076429e+00,
                 2.61076429e-01],
                [ 2.63214286e+00,
                 -2.63214286e-01],
                [ 2.65348000e+00,
                 2.65348000e-01],
                [ 2.67477500e+00,
                 -2.67477500e-01],
                [ 2.69601667e+00,
                 2.69601667e-01],
                [ 2.71720000e+00,
                 -2.71720000e-01],
                [ 2.73831250e+00,
                 2.73831250e-01],
                [ 2.75934286e+00,
                 -2.75934286e-01],
                [ 2.78028000e+00,
                 2.78028000e-01],
                [ 2.80121250e+00,
                 -2.80121250e-01],
                [ 2.82212500e+00,
                 2.82212500e-01],
                [ 2.84300000e+00,
                 -2.84300000e-01],
                [ 2.86382500e+00,
                 2.86382500e-01],
                [ 2.88459000e+00,
                 -2.88459000e-01],
                [ 2.90528571e+00,
                 2.90528571e-01],
                [ 2.92590000e+00,
                 -2.92590000e-01],
                [ 2.94642500e+00,
                 2.94642500e-01],
                [ 2.96685000e+00,
                 -2.96685000e-01],
                [ 2.98716250e+00,
                 2.98716250e-01],
                [ 3.00745000e+00,
                 -3.00745000e-01],
                [ 3.02770000e+00,
                 3.02770000e-01],
                [ 3.04790000e+00,
                 -3.04790000e-01],
                [ 3.06803750e+00,
                 3.06803750e-01],
                [ 3.08810000e+00,
                 -3.08810000e-01],
                [ 3.10807500e+00,
                 3.10807500e-01],
                [ 3.12805000e+00,
                 -3.12805000e-01],
                [ 3.14791250e+00,
                 3.14791250e-01],
                [ 3.16765000e+00,
                 -3.16765000e-01],
                [ 3.18735000e+00,
                 3.18735000e-01],
                [ 3.20700000e+00,
                 -3.20700000e-01],
                [ 3.22659000e+00,
                 3.22659000e-01],
                [ 3.24610000e+00,
                 -3.24610000e-01],
                [ 3.26551250e+00,
                 3.26551250e-01],
                [ 3.28481250e+00,
                 -3.28481250e-01],
                [ 3.30408750e+00,
                 3.30408750e-01],
                [ 3.32331250e+00,
                 -3.32331250e-01],
                [ 3.34247500e+00,
                 3.34247500e-01],
                [ 3.36156250e+00,
                 -3.36156250e-01],
                [ 3.38056250e+00,
                 3.38056250e-01],
                [ 3.39947500e+00,
                 -3.39947500e-01],
                [ 3.41828750e+00,
                 3.41828750e-01],
                [ 3.43700000e+00,
                 -3.43700000e-01],
                [ 3.45560000e+00,
                 3.45560000e-01],
                [ 3.47407500e+00,
                 -3.47407500e-01],
                [ 3.49250000e+00,
                 3.49250000e-01],
                [ 3.51086250e+00,
                 -3.51086250e-01],
                [ 3.52915000e+00,
                 3.52915000e-01],
                [ 3.54735000e+00,
                 -3.54735000e-01],
                [ 3.56545000e+00,
                 3.56545000e-01],
                [ 3.58343750e+00,
                 -3.58343750e-01],
                [ 3.60138750e+00,
                 3.60138750e-01],
                [ 3.61928750e+00,
                 -3.61928750e-01],
                [ 3.63712500e+00,
                 3.63712500e-01],
                [ 3.65488750e+00,
                 -3.65488750e-01],
                [ 3.67256250e+00,
                 3.67256250e-01],
                [ 3.69013750e+00,
                 -3.69013750e-01],
                [ 3.70760000e+00,
                 3.70760000e-01],
                [ 3.72493750e+00,
                 -3.72493750e-01],
                [ 3.74213750e+00,
                 3.74213750e-01],
                [ 3.75928750e+00,
                 -3.75928750e-01],
                [ 3.77628750e+00,
                 3.77628750e-01],
                [ 3.79322500e+00,
                 -3.79322500e-01],
                [ 3.81008750e+00,
                 3.81008750e-01],
                [ 3.82686250e+00,
                 -3.82686250e-01],
                [ 3.84353750e+00,
                 3.84353750e-01],
                [ 3.86010000e+00,
                 -3.86010000e-01],
                [ 3.87653750e+00,
                 3.87653750e-01],
                [ 3.89283750e+00,
                 -3.89283750e-01],
                [ 3.90908750e+00,
                 3.90908750e-01],
                [ 3.92518750e+00,
                 -3.92518750e-01],
                [ 3.94122500e+00,
                 3.94122500e-01],
                [ 3.95718750e+00,
                 -3.95718750e-01],
                [ 3.97306250e+00,
                 3.97306250e-01],
                [ 3.98883750e+00,
                 -3.98883750e-01],
                [ 4.00450000e+00,
                 4.00450000e-01],
                [ 4.02003750e+00,
                 -4.02003750e-01],
                [ 4.03543750e+00,
                 4.03543750e-01],
                [ 4.05070000e+00,
                 -4.05070000e-01],
                [ 4.06581250e+00,
                 4.06581250e-01],
                [ 4.08076250e+00,
                 -4.08076250e-01],
                [ 4.09553750e+00,
                 4.09553750e-01],
                [ 4.11022500e+00,
                 -4.11022500e-01],
                [ 4.12481250e+00,
                 4.12481250e-01],
                [ 4.13928750e+00,
                 -4.13928750e-01],
                [ 4.15363750e+00,
                 4.15363750e-01],
                [ 4.16785000e+00,
                 -4.16785000e-01],
                [ 4.18191250e+00,
                 4.18191250e-01],
                [ 4.19681250e+00,
                 -4.19681250e-01],
                [ 4.21163750e+00,
                 4.21163750e-01],
                [ 4.22628750e+00,
                 -4.22628750e-01],
                [ 4.24075000e+00,
                 4.24075000e-01],
                [ 4.25501250e+00,
                 -4.25501250e-01],
                [ 4.26907500e+00,
                 4.26907500e-01],
                [ 4.28293750e+00,
                 -4.28293750e-01],
                [ 4.29668750e+00,
                 4.29668750e-01],
                [ 4.31031250e+00,
                 -4.31031250e-01],
                [ 4.32381250e+00,
                 4.32381250e-01],
                [ 4.33718750e+00,
                 -4.33718750e-01],
                [ 4.35043750e+00,
                 4.35043750e-01],
                [ 4.36355000e+00,
                 -4.36355000e-01],
                [ 4.37651250e+00,
                 4.37651250e-01],
                [ 4.38931250e+00,
                 -4.38931250e-01],
                [ 4.40193750e+00,
                 4.40193750e-01],
                [ 4.41437500e+00,
                 -4.41437500e-01],
                [ 4.42671250e+00,
                 4.42671250e-01],
                [ 4.43893750e+00,
                 -4.43893750e-01],
                [ 4.45095000e+00,
                 4.45095000e-01],
                [ 4.46283750e+00,
                 -4.46283750e-01],
                [ 4.47458750e+00,
                 4.47458750e-01],
                [ 4.48620000e+00,
                 -4.48620000e-01],
                [ 4.49766250e+00,
                 4.49766250e-01],
                [ 4.50896250e+00,
                 -4.50896250e-01],
                [ 4.52010000e+00,
                 4.52010000e-01],
                [ 4.53107500e+00,
                 -4.53107500e-01],
                [ 4.54188750e+00,
                 4.54188750e-01],
                [ 4.55252500e+00,
                 -4.55252500e-01],
                [ 4.56298750e+00,
                 4.56298750e-01],
                [ 4.57326250e+00,
                 -4.57326250e-01],
                [ 4.58335000e+00,
                 4.58335000e-01],
                [ 4.59323750e+00,
                 -4.59323750e-01],
                [ 4.60292500e+00,
                 4.60292500e-01],
                [ 4.61241250e+00,
                 -4.61241250e-01],
                [ 4.62170000e+00,
                 4.62170000e-01],
                [ 4.63078750e+00,
                 -4.63078750e-01],
                [ 4.63967500e+00,
                 4.63967500e-01],
                [ 4.64836250e+00,
                 -4.64836250e-01],
                [ 4.65685000e+00,
                 4.65685000e-01],
                [ 4.66513750e+00,
                 -4.66513750e-01],
                [ 4.67321250e+00,
                 4.67321250e-01],
                [ 4.68107500e+00,
                 -4.68107500e-01],
                [ 4.68872500e+00,
                 4.68872500e-01],
                [ 4.69616250e+00,
                 -4.69616250e-01],
                [ 4.70338750e+00,
                 4.70338750e-01],
                [ 4.71038750e+00,
                 -4.71038750e-01],
                [ 4.71716250e+00,
                 4.71716250e-01],
                [ 4.72370000e+00,
                 -4.72370000e-01],
                [ 4.72998750e+00,
                 4.72998750e-01],
                [ 4.73601250e+00,
                 -4.73601250e-01],
                [ 4.74177500e+00,
                 4.74177500e-01],
                [ 4.74726250e+00,
                 -4.74726250e-01],
                [ 4.75247500e+00,
                 4.75247500e-01],
                [ 4.75740000e+00,
                 -4.75740000e-01],
                [ 4.76203750e+00,
                 4.76203750e-01],
                [ 4.76637500e+00,
                 -4.76637500e-01],
                [ 4.77041250e+00,
                 4.77041250e-01],
                [ 4.77413750e+00,
                 -4.77413750e-01],
                [ 4.77755000e+00,
                 4.77755000e-01],
                [ 4.78063750e+00,
                 -4.78063750e-01],
                [ 4.78338750e+00,
                 4.78338750e-01],
                [ 4.78580000e+00,
                 -4.78580000e-01],
                [ 4.78786250e+00,
                 4.78786250e-01],
                [ 4.78956250e+00,
                 -4.78956250e-01],
                [ 4.79088750e+00,
                 4.79088750e-01],
                [ 4.79182500e+00,
                 -4.79182500e-01],
                [ 4.79237500e+00,
                 4.79237500e-01],
                [ 4.79253750e+00,
                 -4.79253750e-01],
                [ 4.79230000e+00,
                 4.79230000e-01],
                [ 4.79166250e+00,
                 -4.79166250e-01],
                [ 4.79062500e+00,
                 4.79062500e-01],
                [ 4.78918750e+00,
                 -4.78918750e-01],
                [ 4.78735000e+00,
                 4.78735000e-01],
                [ 4.78511250e+00,
                 -4.78511250e-01],
                [ 4.78247500e+00,
                 4.78247500e-01],
                [ 4.77943750e+00,
                 -4.77943750e-01],
                [ 4.77598750e+00,
                 4.77598750e-01],
                [ 4.77212500e+00,
                 -4.77212500e-01],
                [ 4.76785000e+00,
                 4.76785000e-01],
                [ 4.76316250e+00,
                 -4.76316250e-01],
                [ 4.75805000e+00,
                 4.75805000e-01],
                [ 4.75251250e+00,
                 -4.75251250e-01],
                [ 4.74653750e+00,
                 4.74653750e-01],
                [ 4.74011250e+00,
                 -4.74011250e-01],
                [ 4.73323750e+00,
                 4.73323750e-01],
                [ 4.72590000e+00,
                 -4.72590000e-01],
                [ 4.71810000e+00,
                 4.71810000e-01],
                [ 4.70982500e+00,
                 -4.70982500e-01],
                [ 4.70106250e+00,
                 4.70106250e-01],
                [ 4.69180000e+00,
                 -4.69180000e-01],
                [ 4.68203750e+00,
                 4.68203750e-01],
                [ 4.67176250e+00,
                 -4.67176250e-01],
                [ 4.66107500e+00,
                 4.66107500e-01],
                [ 4.64996250e+00,
                 -4.64996250e-01],
                [ 4.63842500e+00,
                 4.63842500e-01],
                [ 4.62645000e+00,
                 -4.62645000e-01],
                [ 4.61403750e+00,
                 4.61403750e-01],
                [ 4.60117500e+00,
                 -4.60117500e-01],
                [ 4.58786250e+00,
                 4.58786250e-01],
                [ 4.57408750e+00,
                 -4.57408750e-01],
                [ 4.55985000e+00,
                 4.55985000e-01],
                [ 4.54513750e+00,
                 -4.54513750e-01],
                [ 4.53093750e+00,
                 4.53093750e-01],
                [ 4.51625000e+00,
                 -4.51625000e-01],
                [ 4.50106250e+00,
                 4.50106250e-01],
                [ 4.48536250e+00,
                 -4.48536250e-01],
                [ 4.46913750e+00,
                 4.46913750e-01],
                [ 4.45237500e+00,
                 -4.45237500e-01],
                [ 4.43506250e+00,
                 4.43506250e-01],
                [ 4.41718750e+00,
                 -4.41718750e-01],
                [ 4.39873750e+00,
                 4.39873750e-01],
                [ 4.37970000e+00,
                 -4.37970000e-01],
                [ 4.36006250e+00,
                 4.36006250e-01],
                [ 4.34081250e+00,
                 -4.34081250e-01],
                [ 4.32093750e+00,
                 4.32093750e-01],
                [ 4.30142500e+00,
                 -4.30142500e-01],
                [ 4.28126250e+00,
                 4.28126250e-01],
                [ 4.26143750e+00,
                 -4.26143750e-01],
                [ 4.24093750e+00,
                 4.24093750e-01],
                [ 4.22075000e+00,
                 -4.22075000e-01],
                [ 4.20086250e+00,
                 4.20086250e-01],
                [ 4.18026250e+00,
                 -4
```

```

[ 0.09421488e-01]],

[[ 5.47708490e+00],
 [-5.47708490e-01]],

[[ 4.48125128e+00],
 [ 4.48125128e-01]],

[[ 3.66647832e+00],
 [-3.66647832e-01]],

[[ 2.99984590e+00],
 [ 2.99984590e-01]],

[[ 2.45441937e+00],
 [-2.45441937e-01]],

[[ 2.00816130e+00],
 [ 2.00816130e-01]],

[[ 1.64304107e+00],
 [-1.64304107e-01]],

[[ 1.34430633e+00],
 [ 1.34430633e-01]],

[[ 1.09988700e+00],
 [-1.09988700e-01]],

[[ 8.99907542e-01],
 [ 8.99907542e-02]],

[[ 7.36287989e-01],
 [-7.36287989e-02]],

[[ 6.02417445e-01],
 [ 6.02417445e-02]],

[[ 4.92887001e-01],
 [-4.92887001e-02]],

[[ 4.03271182e-01],
 [ 4.03271182e-02]],

[[ 3.29949149e-01],
 [-3.29949149e-02]],

[[ 2.69958395e-01],
 [ 2.69958395e-02]],

[[ 2.20875050e-01],
 [-2.20875050e-02]],

[[ 1.80715950e-01],
 [ 1.80715950e-02]],

[[ 1.47858505e-01],
 [-1.47858505e-02]],

[[ 1.20975140e-01],
 [ 1.20975140e-02]],

[[ 9.89796602e-02],
 [-9.89796602e-03]],

```

```

[[ 8.09833583e-02],
 [ 8.09833583e-03]],

[[ 6.62591114e-02],
 [-6.62591114e-03]],

[[ 5.42120002e-02],
 [ 5.42120002e-03]],

[[ 4.43552729e-02],
 [-4.43552729e-03]],

[[ 3.62906778e-02],
 [ 3.62906778e-03]],

[[ 2.96923728e-02],
 [-2.96923728e-03]],

[[ 2.42937595e-02],
 [ 2.42937595e-03]]])

```

In [41]:

```

#define ranges for x1 and x2
x1 = np.linspace(-12, 12, 2000)
x2 = np.linspace(-5, 5, 2000)

#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)
X1, X2 = np.meshgrid(x1, x2)

#function to calculate f1 value
#x1, x2: inputs
#gamma: function input
def f1_plot(x1, x2, gamma = 10):
    return 0.5 * ((x1 ** 2) + (gamma * (x2 ** 2)))

#given points on the contour grid, get function values
F = f1_plot(X1, X2)

#set figure size
plt.figure(figsize = (12, 12))

#plot f1 exact line search iterates
for point in f1_exact_line_search_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label  $x^{(0)}$  and  $x^{(1)}$ 
offset = 0.4
plt.text(f1_exact_line_search_iterates[0][0, 0] + offset, f1_exact_line_search_
plt.text(f1_exact_line_search_iterates[1][0, 0] + 0.4 * offset, f1_exact_line_s

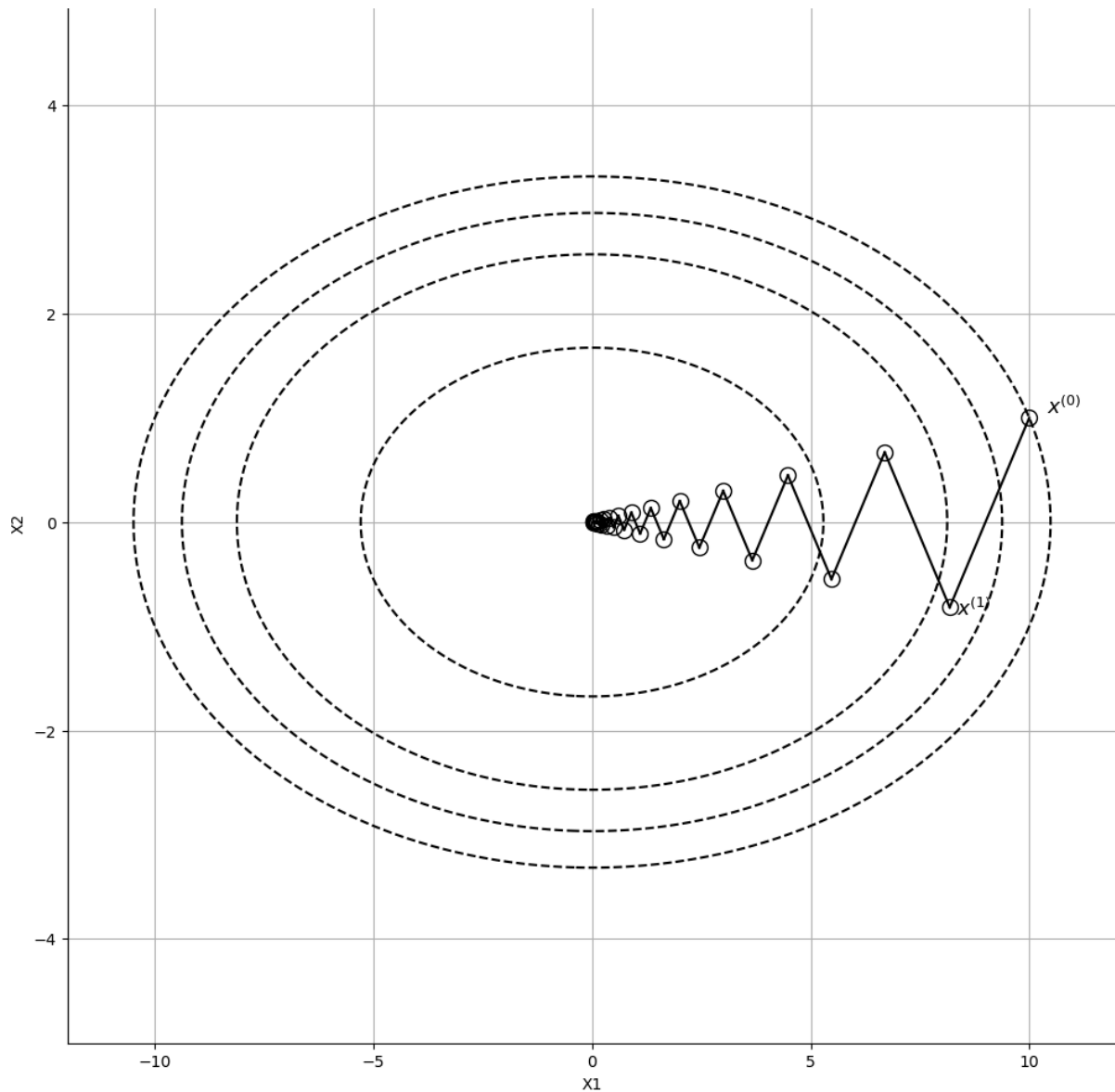
#Connect iterates via a line
for i in range(len(f1_exact_line_search_iterates)-1):
    plt.plot([f1_exact_line_search_iterates[i][0, 0], f1_exact_line_search_iter

#Plot contour lines
plt.contour(X1, X2, F, levels = [14, 33, 44, 55], colors='black', linestyle='d
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Quadratic Function[Exact Line Search]')
plt.grid(True)
plt.show()

```

Contour Plot of Quadratic Function[Exact Line Search]





In [42]: `#run gradient descent, with backtracking algorithm, for f1(i.e. the quadratic f`  
`f1_backtracking_iterates, f1_backtracking_function_values = backtracking_line_s`

In [43]: `#define ranges for x1 and x2`  
`x1 = np.linspace(-12, 12, 2000)`  
`x2 = np.linspace(-5, 5, 2000)`  
  
`#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)`  
`X1, X2 = np.meshgrid(x1, x2)`  
  
`#function to calculate f1 value`  
`#x1, x2: inputs`  
`#gamma: function input`  
`def f1_plot(x1, x2, gamma = 10):`  
 `return 0.5 * ((x1 ** 2) + (gamma * (x2 ** 2)))`  
  
`#given points on the contour grid, get function values`  
`F = f1_plot(X1, X2)`  
  
`#set figure size`  
`plt.figure(figsize = (12, 12))`  
  
`#plot f1 backtracking iterates`

```

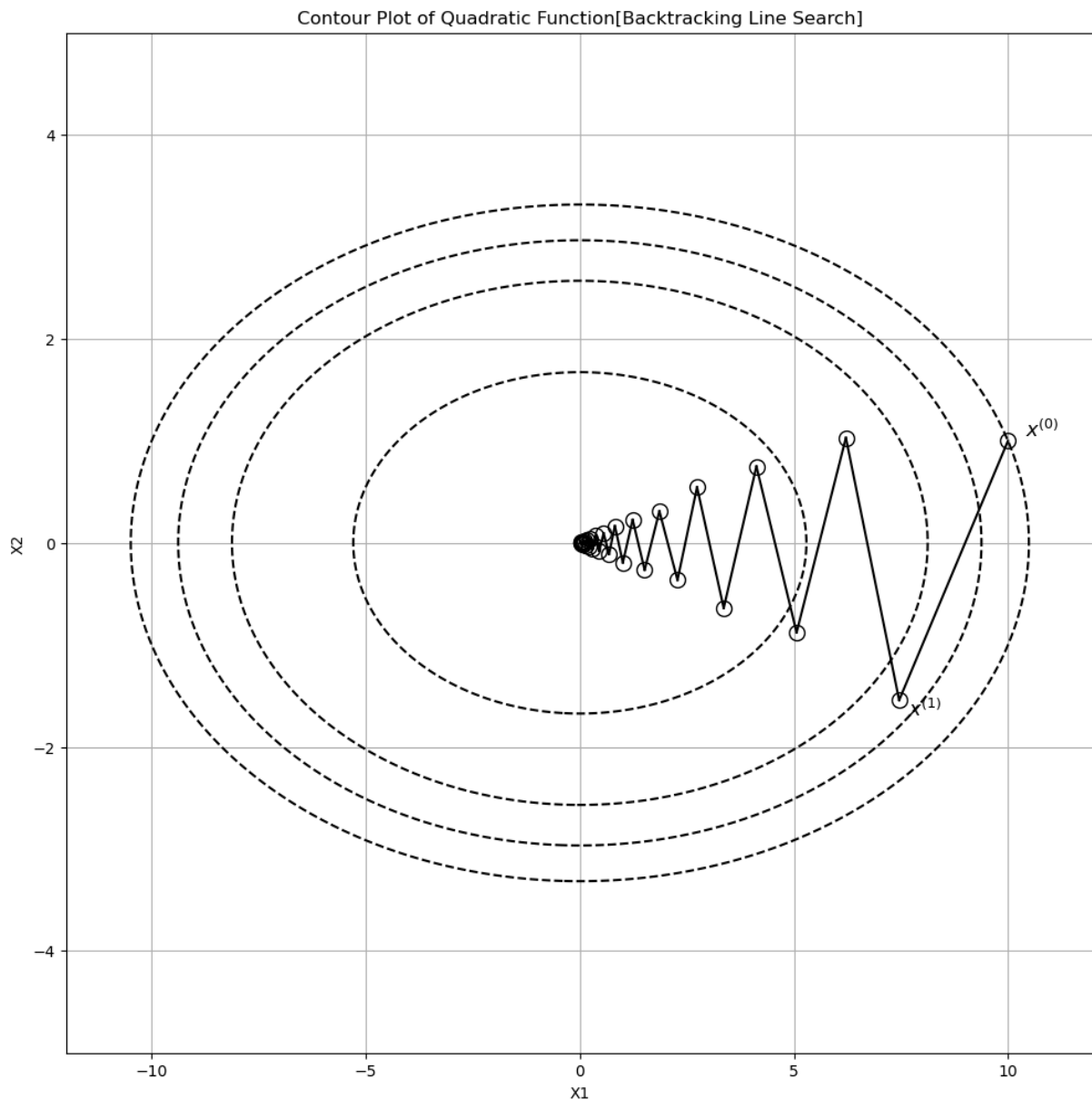
#plot f1 backtracking iterates
for point in f1_backtracking_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label  $x^{(0)}$  and  $x^{(1)}$ 
offset = 0.4
plt.text(f1_backtracking_iterates[0][0, 0] + offset, f1_backtracking_iterates[0
plt.text(f1_backtracking_iterates[1][0, 0] + 0.5 * offset, f1_backtracking_iter

#Connect iterates via a line
for i in range(len(f1_backtracking_iterates)-1):
    plt.plot([f1_backtracking_iterates[i][0, 0], f1_backtracking_iterates[i+1][

#Plot contour lines
plt.contour(X1, X2, F, levels = [14, 33, 44, 55], colors='black', linestyle='d
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Quadratic Function[Backtracking Line Search]')
plt.grid(True)
plt.show()

```



```

In [44]: #optimal function value
          n_star = 0

```

```

#compute error values(i.e. difference between function value and optimal functi
f1_exact_line_search_distance_from_optimal = f1_exact_line_search_function_valu
f1_backtracking_distance_from_optimal = f1_backtracking_function_values - p_star
f1_vanilla_gd_distance_from_optimal = f1_vanilla_gd_function_values - p_star

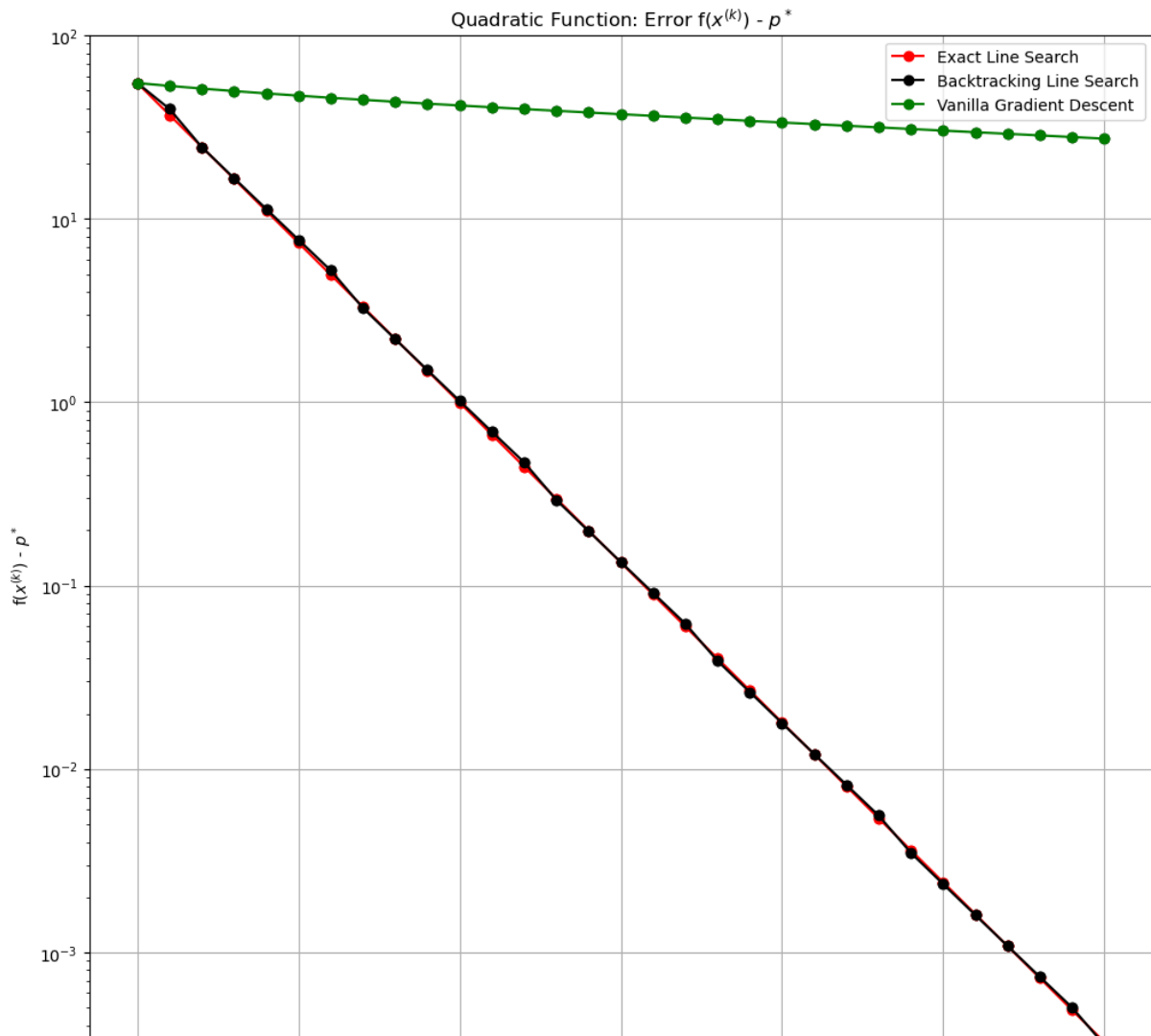
#set figure size
plt.figure(figsize = (12, 12))

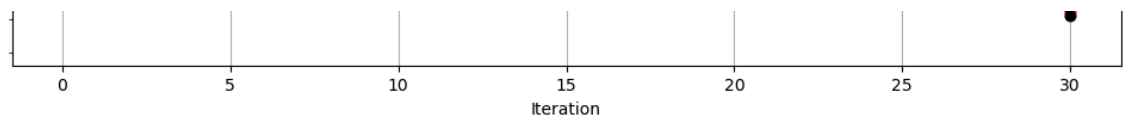
#plot error values
line1, = plt.plot(f1_exact_line_search_distance_from_optimal, marker='o', color=
line2, = plt.plot(f1_backtracking_distance_from_optimal, marker='o', color='bla
line3, = plt.plot(f1_vanilla_gd_distance_from_optimal, marker='o', color='green

#indicate, via points, each error value on the lines
plt.scatter(range(len(f1_exact_line_search_distance_from_optimal)), f1_exact_li
plt.scatter(range(len(f1_backtracking_distance_from_optimal)), f1_backtracking_
plt.scatter(range(len(f1_vanilla_gd_distance_from_optimal)), f1_vanilla_gd_dist

#label axes, label title, and add legend
plt.xlabel('Iteration')
plt.ylabel(r' $f(x^{(k)}) - p^*$ ')
plt.yscale('log')
plt.title(r'Quadratic Function: Error  $f(x^{(k)}) - p^*$ ')
plt.legend(handles=[line1, line2, line3], labels=['Exact Line Search', 'Backtra
plt.grid(True)
plt.show()

```





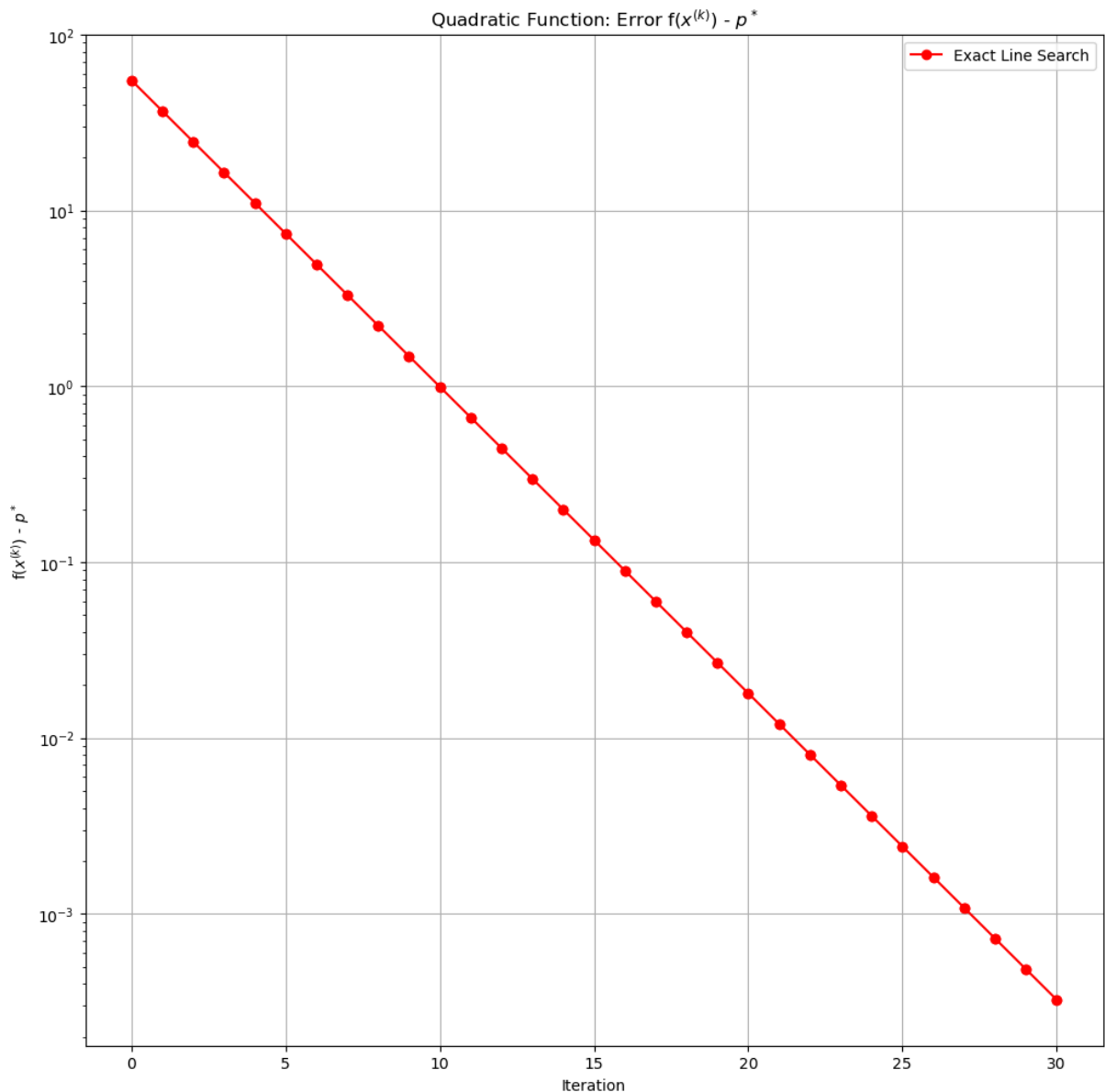
In [45]:

```
#set figure size
plt.figure(figsize = (12, 12))

#plot error values
line1, = plt.plot(f1_exact_line_search_distance_from_optimal, marker='o', color

#indicate, via points, each error value on the lines
plt.scatter(range(len(f1_exact_line_search_distance_from_optimal)), f1_exact_li

#label axes, label title, and add legend
plt.xlabel('Iteration')
plt.ylabel(r' $f(x^{(k)}) - p^*$ ')
plt.yscale('log')
plt.title(r'Quadratic Function: Error  $f(x^{(k)}) - p^*$ ')
plt.legend(handles=[line1], labels=['Exact Line Search'])
plt.grid(True)
plt.show()
```



In [46]:

```

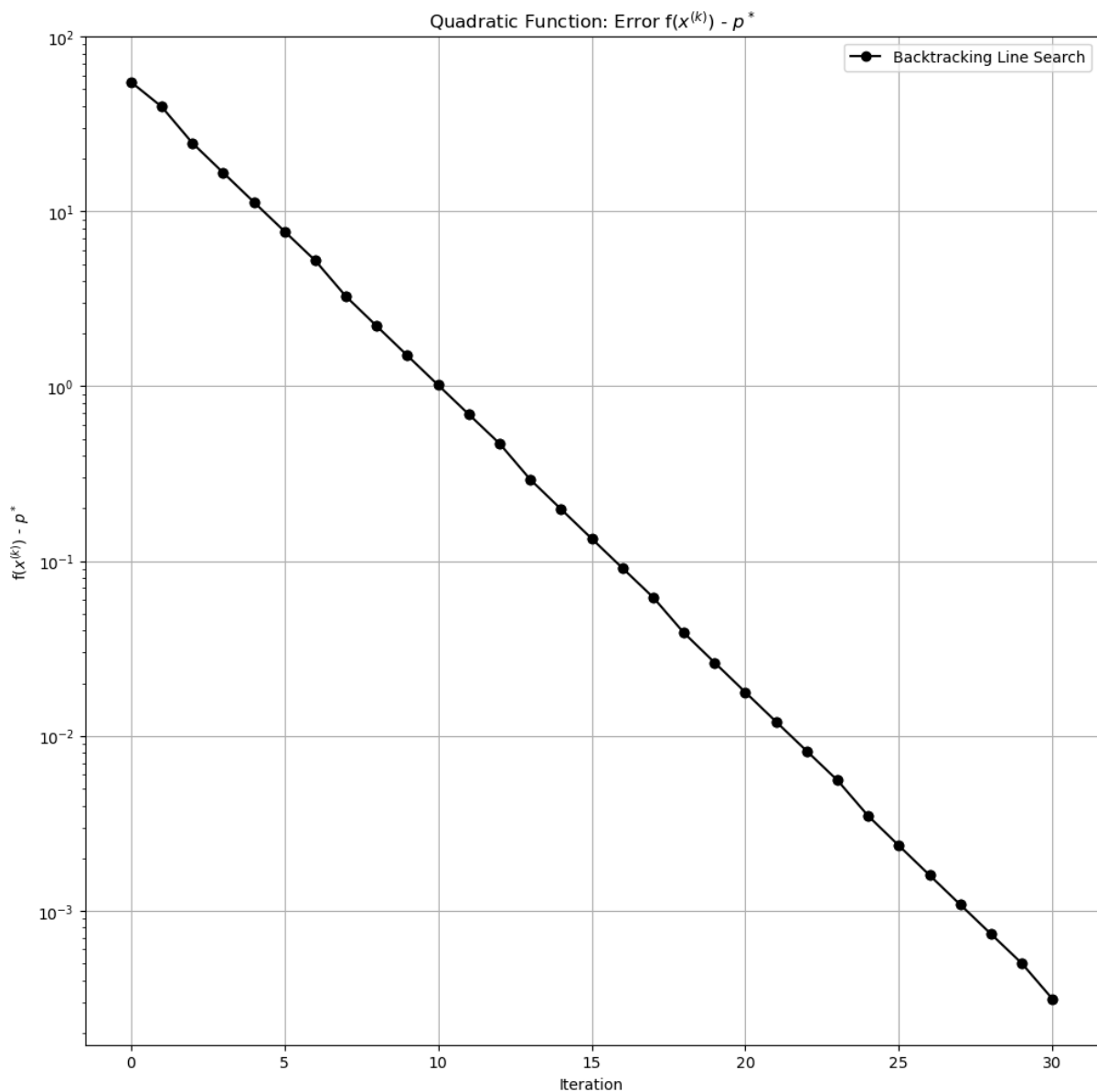
#set figure size
plt.figure(figsize = (12, 12))

#plot error values
line2, = plt.plot(f1_backtracking_distance_from_optimal, marker='o', color='bla

#indicate, via points, each error value on the lines
plt.scatter(range(len(f1_backtracking_distance_from_optimal)), f1_backtracking_

#label axes, label title, and add legend
plt.xlabel('Iteration')
plt.ylabel(r' $f(x^{(k)}) - p^*$ ')
plt.yscale('log')
plt.title(r'Quadratic Function: Error  $f(x^{(k)}) - p^*$ ')
plt.legend(handles=[line2], labels=['Backtracking Line Search'])
plt.grid(True)
plt.show()

```



In [47]:

```
f1_exact_line_search_function_values
```

```
Out[47]: array([5.50000000e+01, 3.68181818e+01, 2.46468820e+01, 1.64991524e+01,
1.10448872e+01, 7.39368480e+00, 4.94914800e+00, 3.31379500e+00,
```

```
1.20110072e+01, 7.33333333e+00, 1.51515151e+00, 3.33333333e+00,
2.21799150e+00, 1.48477117e+00, 9.93937726e-01, 6.65363271e-01,
4.45408471e-01, 2.98166001e-01, 1.99598728e-01, 1.33615677e-01,
8.94452056e-02, 5.98765426e-02, 4.00826442e-02, 2.68321833e-02,
1.79620401e-02, 1.20241756e-02, 8.04924150e-03, 5.38833522e-03,
3.60706738e-03, 2.41464841e-03, 1.61641753e-03, 1.08206463e-03,
7.24357313e-04, 4.84900350e-04, 3.24602714e-04])
```

In [48]: `f1_backtracking_function_values`

Out[48]: `array([5.50000000e+01, 3.96986338e+01, 2.46086126e+01, 1.66722492e+01,
1.13123758e+01, 7.68772777e+00, 5.23308749e+00, 3.26409747e+00,
2.20882200e+00, 1.49687126e+00, 1.01593663e+00, 6.90619922e-01,
4.70254563e-01, 2.92917773e-01, 1.98269155e-01, 1.34399184e-01,
9.12435981e-02, 6.20446346e-02, 3.88804678e-02, 2.62874400e-02,
1.77979768e-02, 1.20678786e-02, 8.19523408e-03, 5.57434147e-03,
3.48855955e-03, 2.35923403e-03, 1.59774461e-03, 1.08364628e-03,
7.36111651e-04, 5.00849565e-04, 3.13025760e-04])`

## Non-Quadratic Function

$$f_2(x_1, x_2) = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}$$

$$\frac{\partial f_2}{\partial x_1} = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1}$$

$$\frac{\partial f_2}{\partial x_2} = 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1}$$

$$\nabla f_2(x) = \begin{pmatrix} e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1} \\ 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} \end{pmatrix}$$

Exact Line Search:

$$f_2(x_1 + \alpha\Delta x_1, x_2 + \alpha\Delta x_2) = e^{x_1+\alpha\Delta x_1+3x_2+3\alpha\Delta x_2-0.1} + e^{x_1+\alpha\Delta x_1-3x_2-3\alpha\Delta x_2-0.1} + e^{-x_1-\alpha\Delta x_1}$$

$$\frac{d}{d\alpha} f_2(x_1 + \alpha\Delta x_1, x_2 + \alpha\Delta x_2) = (\Delta x_1 + 3\Delta x_2)e^{x_1+\alpha\Delta x_1+3x_2+3\alpha\Delta x_2-0.1} + (\Delta x_1 - 3\Delta x_2)e^{x_1+\alpha\Delta x_1-3x_2-3\alpha\Delta x_2-0.1} - e^{-x_1-\alpha\Delta x_1}$$

```
In [49]: #Calculate the value of the non-quadratic function at x
def f2(x: np.ndarray):
    x1 = x[0, 0]
    x2 = x[1, 0]

    A = np.exp(x1 + (3 * x2) - 0.1)
    B = np.exp(x1 - (3 * x2) - 0.1)
    C = np.exp((-1 * x1) - 0.1)

    return A + B + C

#Calculate the value of the gradient of the non-quadratic function at x
def f2_gradient(x: np.ndarray):
    x1 = x[0, 0]
    x2 = x[1, 0]

    A = x1 + (3 * x2) - 0.1
    B = x1 - (3 * x2) - 0.1
    C = (-1 * x1) - 0.1

    gradient_vector = np.zeros(shape = x.shape)
    gradient_vector[0, 0] = np.exp(A) + np.exp(B) - np.exp(C)
    gradient_vector[1, 0] = (3 * np.exp(A)) - (3 * np.exp(B))
```

```

    return gradient_vector

#Since there is no exact form for line search for the non-quadratic function, r
#x: current iterate
#delta_x: descent direction
def f2_exact_line_search(x: np.ndarray, delta_x: np.ndarray):
    grid_search_alphas = np.arange(start = 0, stop = 1, step = 0.0003) #alphas

    #store best alpha and best function value for that alpha
    best_alpha = None
    best_function_value = np.inf

    #run a grid search over alphas
    for alpha in grid_search_alphas:
        #if we have found a better alpha
        if f2(x + (alpha * delta_x)) < best_function_value:
            best_alpha = alpha #update best alpha
            best_function_value = f2(x + (alpha * delta_x)) #update best alpha

    return best_alpha

#Run the backtracking algorithm for the optimization problem involving the non-
#x: current iterate
#delta_x: descent direction
#alpha: value of alpha in backtracking algorithm
#beta: value of beta in backtracking algorithm
def f2_backtracking_algorithm(x: np.ndarray, delta_x: np.ndarray, alpha = 0.1,
                             t = 1):
    while (f2(x + (t * delta_x))) > (f2(x) + (alpha * t * (f2_gradient(x).T @ delta_x))):
        t = beta * t

    return t

```

```
In [50]: x0 = np.array([[ -0.9], [ 1.1]]) #initial iterate for gradient descent
```

```
In [51]: #run vanilla gradient descent for f2(i.e. the non-quadratic function)
f2_vanilla_gd_iterates, f2_vanilla_gd_function_values = vanilla_gradient_descent(x0, delta_x)
```

```
In [52]: #run gradient descent, with exact line search, for f2(i.e. the non-quadratic function)
f2_exact_line_search_iterates, f2_exact_line_search_function_values = exact_line_search(x0, delta_x)
```

```
In [53]: #run gradient descent, with backtracking, for f2(i.e. the non-quadratic function)
f2_backtracking_iterates, f2_backtracking_function_values = backtracking_line_search(x0, delta_x)
```

```
In [54]: #define ranges for x1 and x2
x1 = np.linspace(-3, 2, 2000)
x2 = np.linspace(-1.5, 1.5, 2000)

#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)
X1, X2 = np.meshgrid(x1, x2)

#function to calculate f2 value
#x1, x2: inputs
def f2_plot(x1, x2):
    A = np.exp(x1 + (3 * x2) - 0.1)

```

```

B = np.exp(x1 - (3 * x2) - 0.1)
C = np.exp((-1 * x1) - 0.1)

return A + B + C

#given points on the contour grid, get function values
F = f2_plot(X1, X2)

#set figure size
plt.figure(figsize = (12, 12))

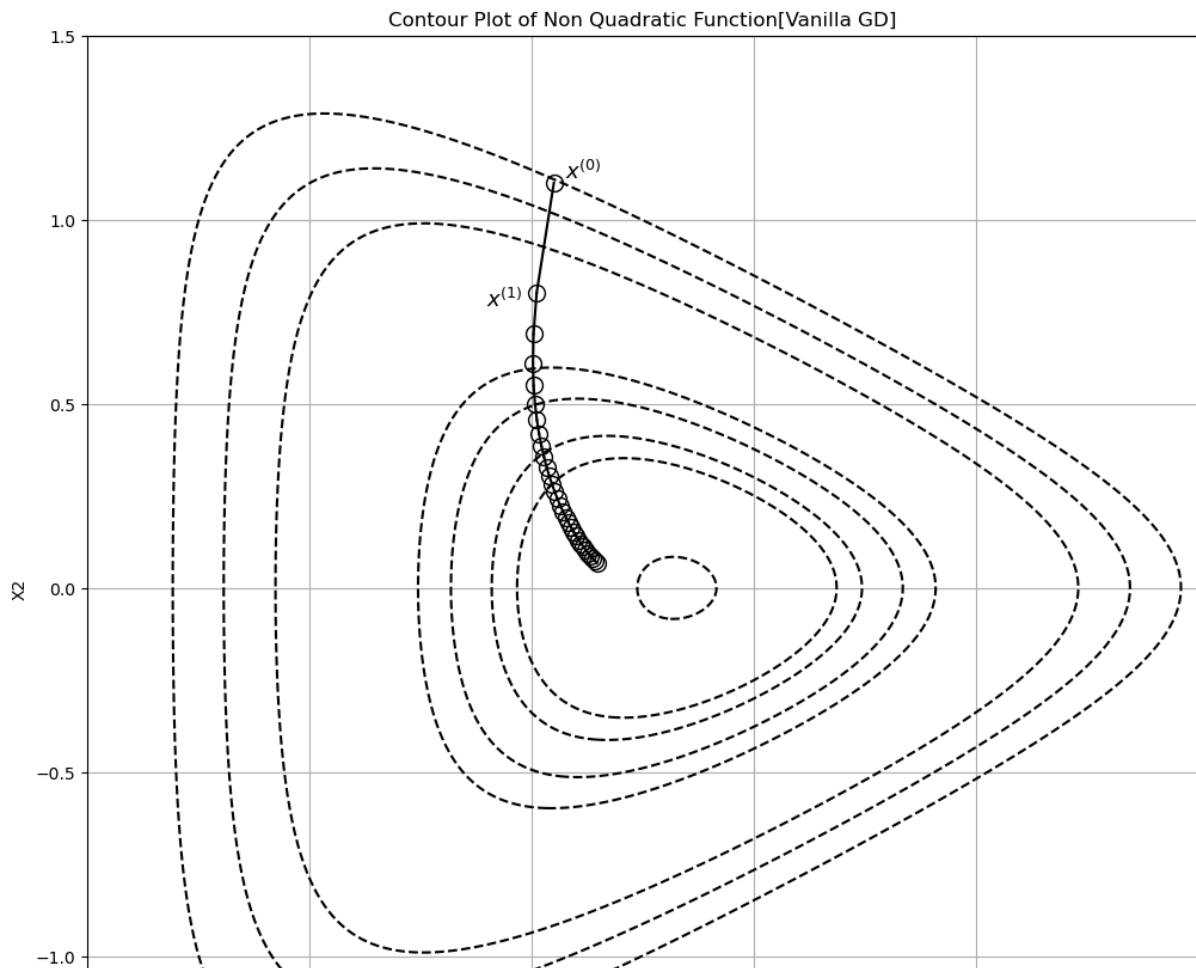
#plot f2 vanilla GD iterates
for point in f2_vanilla_gd_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label  $x^{(0)}$  and  $x^{(1)}$ 
offset = 0.05
plt.text(f2_vanilla_gd_iterates[0][0, 0] + offset, f2_vanilla_gd_iterates[0][1,
plt.text(f2_vanilla_gd_iterates[1][0, 0] - 4.5 * offset, f2_vanilla_gd_iterates

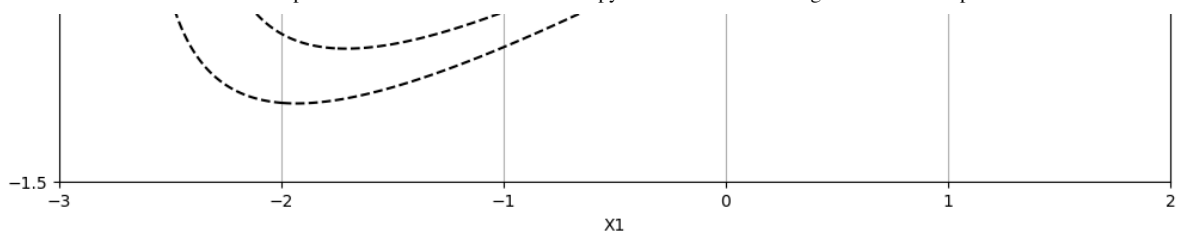
#Connect iterates via a line
for i in range(len(f2_vanilla_gd_iterates)-1):
    plt.plot([f2_vanilla_gd_iterates[i][0, 0], f2_vanilla_gd_iterates[i+1][0, 0

#Plot contours
contour = plt.contour(X1, X2, F, levels = [2.6, 3.25, 3.5, 4, 4.5, 8, 10, 12.5])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Non Quadratic Function[Vanilla GD]')
plt.grid(True)
plt.show()

```







In [55]:

```
#define ranges for x1 and x2
x1 = np.linspace(-3, 2, 2000)
x2 = np.linspace(-1.5, 1.5, 2000)

#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)
X1, X2 = np.meshgrid(x1, x2)

#function to calculate f2 value
#x1, x2: inputs
def f2_plot(x1, x2):
    A = np.exp(x1 + (3 * x2) - 0.1)
    B = np.exp(x1 - (3 * x2) - 0.1)
    C = np.exp((-1 * x1) - 0.1)

    return A + B + C

#given points on the contour grid, get function values
F = f2_plot(X1, X2)

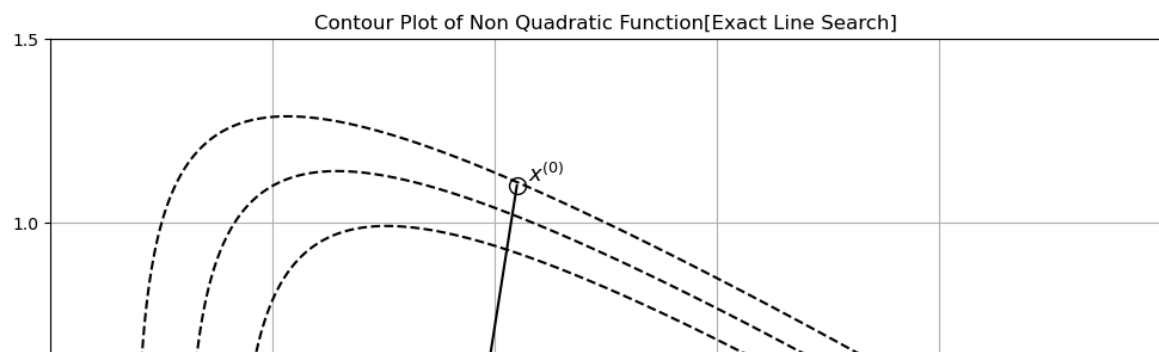
#set figure size
plt.figure(figsize = (12, 12))

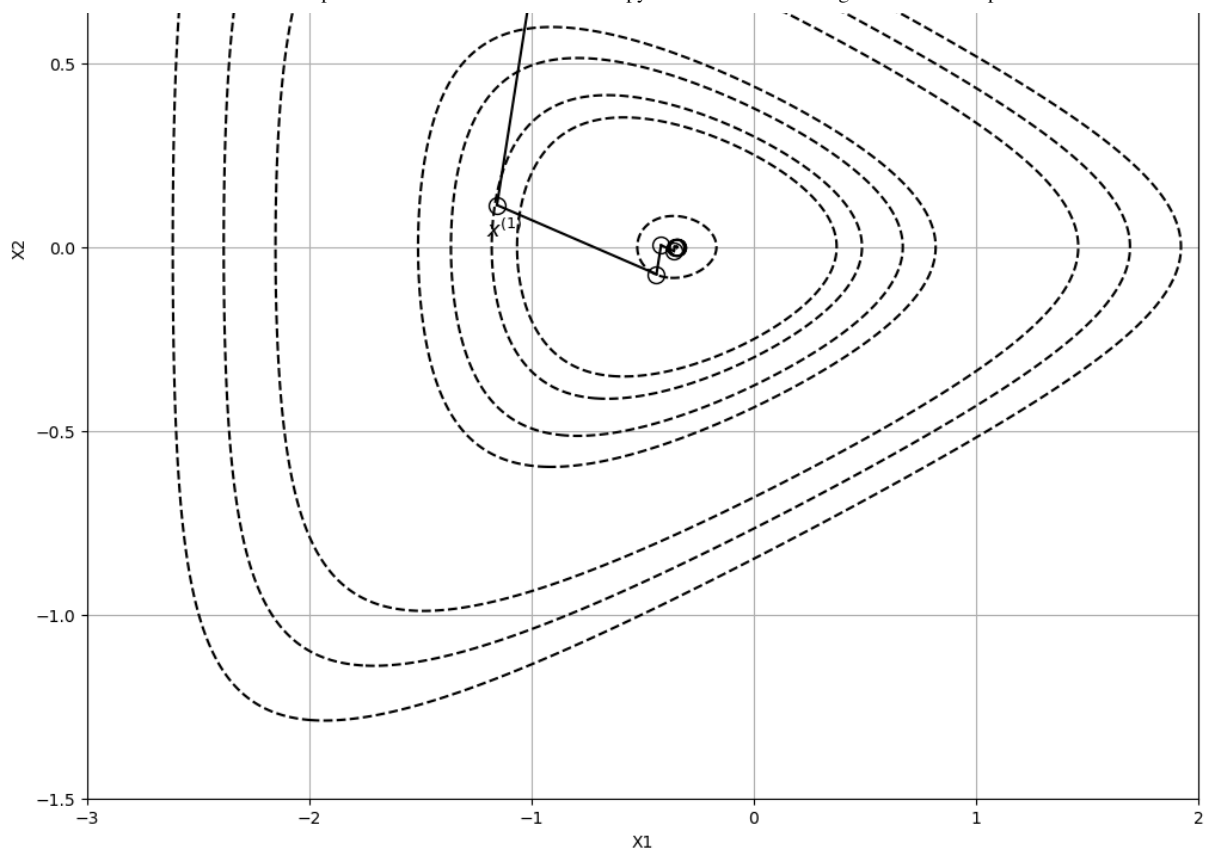
#plot f2 exact line search iterates
for point in f2_exact_line_search_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label  $x^{(0)}$  and  $x^{(1)}$ 
offset = 0.05
plt.text(f2_exact_line_search_iterates[0][0, 0] + offset, f2_exact_line_search_
plt.text(f2_exact_line_search_iterates[1][0, 0] - offset, f2_exact_line_search_

#Connect iterates via a line
for i in range(len(f2_exact_line_search_iterates)-1):
    plt.plot([f2_exact_line_search_iterates[i][0, 0], f2_exact_line_search_iter

#Plot contours
contour = plt.contour(X1, X2, F, levels = [2.6, 3.25, 3.5, 4, 4.5, 8, 10, 12.5])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Non Quadratic Function[Exact Line Search]')
plt.grid(True)
plt.show()
```





```
In [56]: #Print levels of contour plot for debugging information
levels = contour.levels
print("Levels of the contour plot:", levels)
```

Levels of the contour plot: [ 2.6 3.25 3.5 4. 4.5 8. 10. 12.5 ]

```
In [57]: #define ranges for x1 and x2
x1 = np.linspace(-3, 2, 2000)
x2 = np.linspace(-1.5, 1.5, 2000)

#generate (x,y) coordinates for contour grid given x1(x coord) and x2(y coord)
X1, X2 = np.meshgrid(x1, x2)

#function to calculate f2 value
#x1, x2: inputs
def f2_plot(x1, x2):
    A = np.exp(x1 + (3 * x2) - 0.1)
    B = np.exp(x1 - (3 * x2) - 0.1)
    C = np.exp((-1 * x1) - 0.1)

    return A + B + C

#given points on the contour grid, get function values
F = f2_plot(X1, X2)

#set figure size
plt.figure(figsize = (12, 12))

#plot f2 backtracking iterates
for point in f2_backtracking_iterates:
    plt.plot(point[0, 0], point[1, 0], 'o', markersize=10, color='black', marke

#label x^(0) and x^(1)
offset = 0.05
```

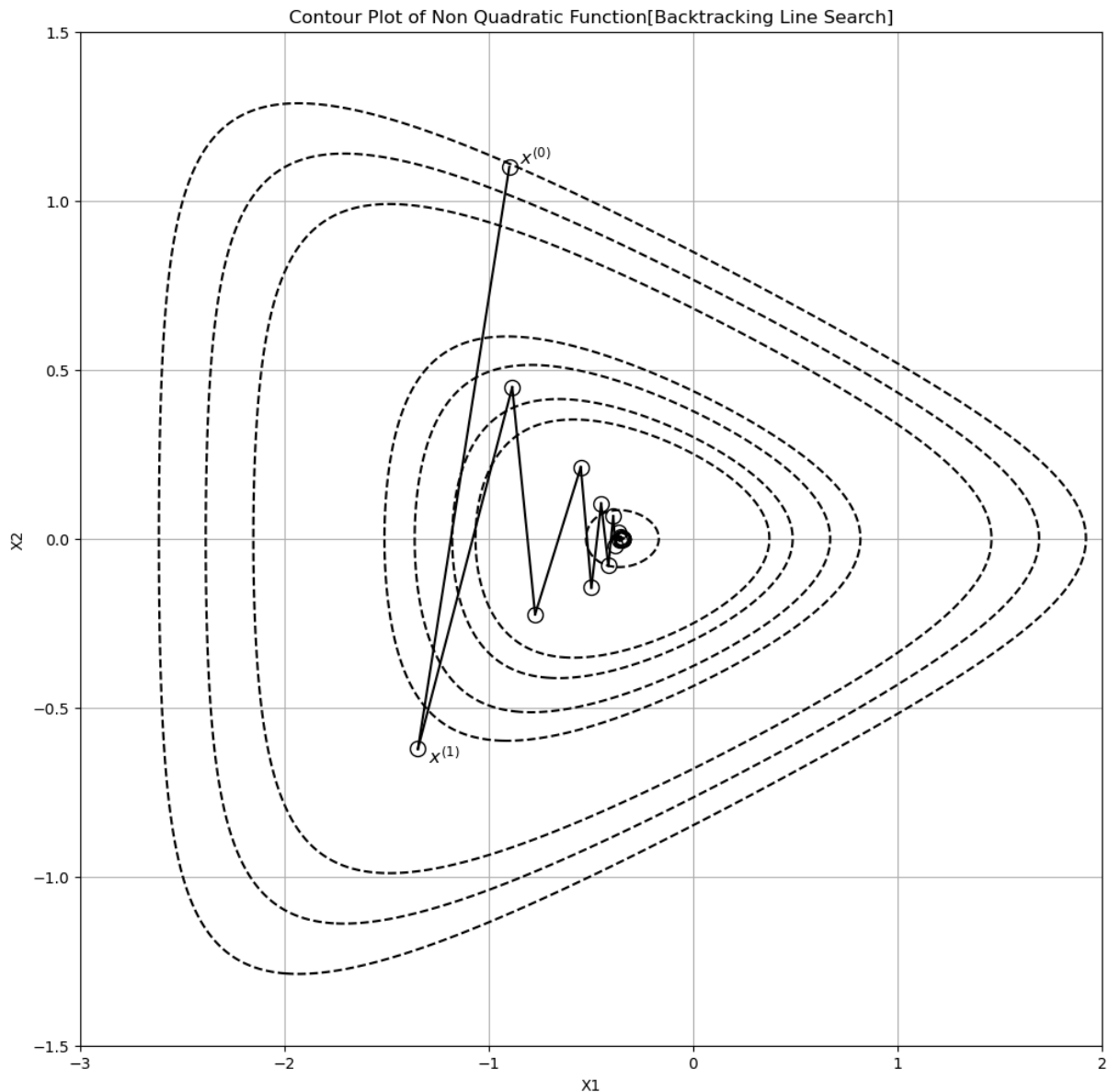
```

plt.text(f2_backtracking_iterates[0][0, 0] + offset, f2_backtracking_iterates[0]
plt.text(f2_backtracking_iterates[1][0, 0] + offset, f2_backtracking_iterates[1]

#Connect iterates via a line
for i in range(len(f2_backtracking_iterates)-1):
    plt.plot([f2_backtracking_iterates[i][0, 0], f2_backtracking_iterates[i+1][

#Plot contours
plt.contour(X1, X2, F, levels = [2.6, 3.25, 3.5, 4, 4.5, 8, 10, 12.5], colors='
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Contour Plot of Non Quadratic Function[Backtracking Line Search]')
plt.grid(True)
plt.show()

```



Calculating the Optimal Value of this Function:

$$f_2(x_1, x_2) = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}$$

$$\frac{\partial f_2}{\partial x_1} = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1}$$

$$\frac{\partial f_2}{\partial x_2} = 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1}$$

$$\nabla f_2(x) = \begin{pmatrix} e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1} \\ 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} \end{pmatrix}$$

Set this gradient to 0

Let's start with the partial derivative with respect to  $x_2$ , which must equal 0 in the gradient:

$$3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} = 0$$

Dividing entire equation by 3:

$$e^{x_1+3x_2-0.1} - e^{x_1-3x_2-0.1} = 0$$

$$e^{x_1+3x_2-0.1} = e^{x_1-3x_2-0.1}$$

$$x_1 + 3x_2 - 0.1 = x_1 - 3x_2 - 0.1$$

It is clear to see that  $x_2 = 0$  must be true.

Now, let's move onto the partial derivative with respect to  $x_1$

$$e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1} = 0$$

Substituting  $x_2 = 0$  gives us:

$$e^{x_1-0.1} + e^{x_1-0.1} - e^{-x_1-0.1} = 0$$

$$2e^{x_1-0.1} - e^{-x_1-0.1} = 0$$

$$2e^{x_1-0.1} = e^{-x_1-0.1}$$

$$e^{(\ln 2)e^{x_1-0.1}} = e^{-x_1-0.1}$$

$$e^{(\ln 2)+x_1-0.1} = e^{-x_1-0.1}$$

$$(\ln 2) + x_1 - 0.1 = -x_1 - 0.1$$

$$2x_1 = -(\ln 2)$$

$$x_1 = \frac{-\ln 2}{2}$$

In [58]:

```
#optimal function value
p_star = f2_plot(-0.5 * np.log(2), 0)

#set figure size
plt.figure(figsize = (12, 12))

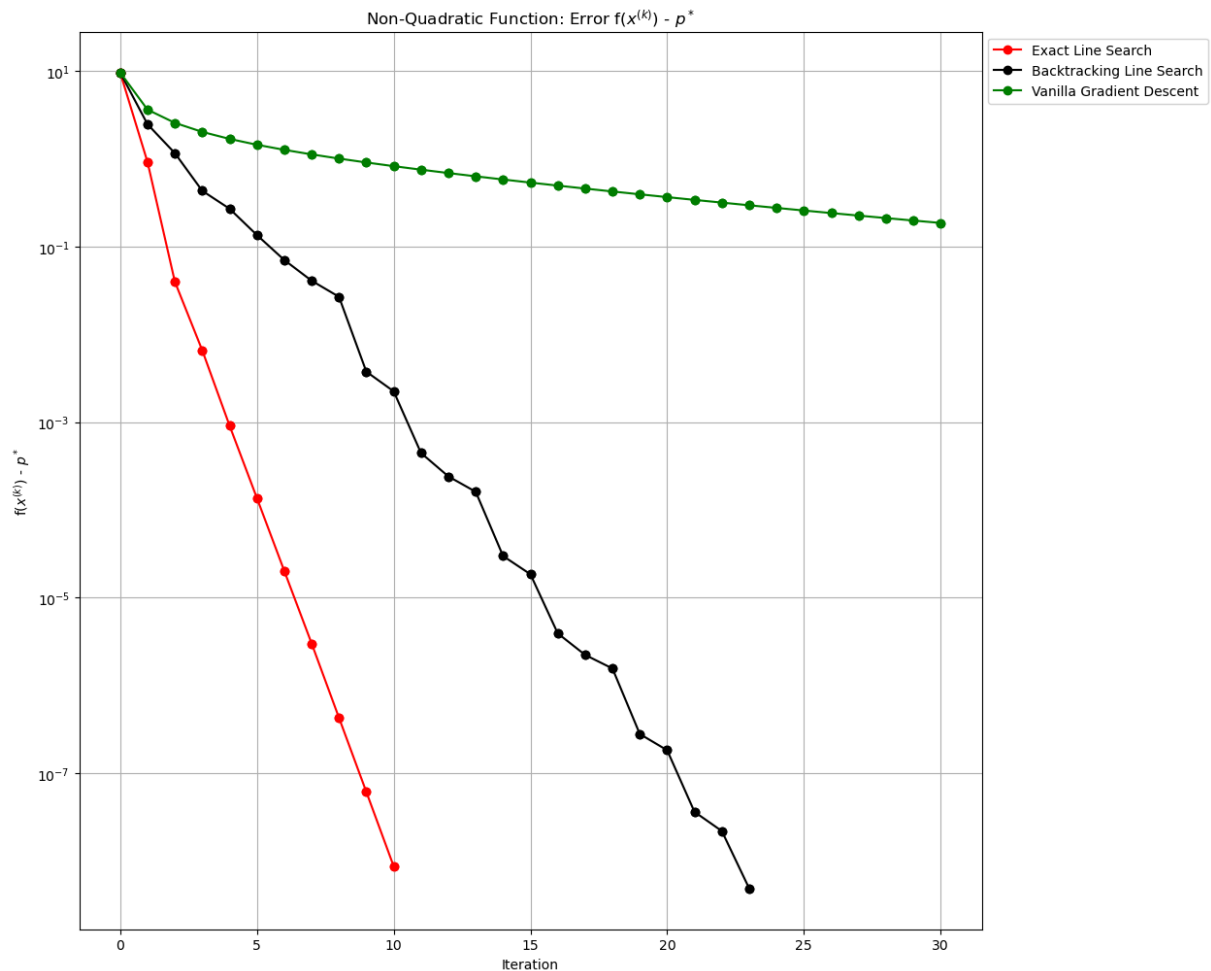
#compute error values(i.e. difference between function value and optimal functi
f2_exact_line_search_distance_from_optimal = f2_exact_line_search_function_valu
f2_backtracking_distance_from_optimal = f2_backtracking_function_values - p_star
f2_vanilla_gd_distance_from_optimal = f2_vanilla_gd_function_values - p_star

#plot error values
line1, = plt.plot(f2_exact_line_search_distance_from_optimal, marker='o', color=
line2, = plt.plot(f2_backtracking_distance_from_optimal, marker='o', color='bla
line3, = plt.plot(f2_vanilla_gd_distance_from_optimal, marker='o', color='green

#indicate, via points, each error value on the lines
```

```
plt.scatter(range(len(f2_exact_line_search_distance_from_optimal)), f2_exact_li
plt.scatter(range(len(f2_backtracking_distance_from_optimal)), f2_backtracking
plt.scatter(range(len(f2_vanilla_gd_distance_from_optimal)), f2_vanilla_gd_dist

#label axes, label title, and add legend
plt.xlabel('Iteration')
plt.ylabel(r'$f(x^{(k)}) - p^*$')
plt.yscale('log')
plt.title(r'Non-Quadratic Function: Error $f(x^{(k)}) - p^*$')
plt.legend(handles=[line1, line2, line3], labels=['Exact Line Search', 'Backtra
plt.grid(True)
plt.show()
```



```
In [59]: p_star = f2_plot(-0.5 * np.log(2), 0)
print(f"Optimal Value of f2 = {p_star}")
```

Optimal Value of f2 = 2.5592666966582156

```
In [60]: ###Print function value at initial value of iterate
print(f"Initial value of f2: {f2(x0)}")
```

Initial value of f2: 12.213291942319392

ConvexOptimization / Homework #4 - Part 2.ipynb

[↑ Top](#)

Preview

Code

Blame

Raw





